CP/M Primer

a most sophisticated operating system

Dr. John F. Stewart
University of Miami
Box 248237
Coral Gables FL 33156

CYBERNETICS INC
8041 NEWMAN AVENUE • SUITE 208
HUNTINGTON BEACH, CALIFORNIA 92647
(714) 848-1922

Reprinted from April 1978 Kilobaud Magazine.

Dr. John F. Stewart
University of Miami
Box 248237
Coral Cables FL 33156

CP/M Primer

a most sophisticated operating system

The author works on an Imsai 8080 in the University of Miami's Hertz Computer Lab.

Many fine articles have appeared in *Kilobaud* describing the principles of operating systems, but, as yet, no one has taken it on himself to present a detailed description of one of the several commercially available operating systems. In this article, we will take a look at the disk-based CP/M system written by Digital Research. In particular, we will look at the version of CP/M that is currently available on the Imsai 8080 microcomputer system. There are only minor differences between the Imsai version and the original, so most of the following will apply to both.

The CP/M operating system is currently available from Digital Research for $70, including documentation and system diskette. I estimate it would take three to six man-months for a sophisticated programmer to produce an operating system with CP/M capabilities. Thus, one of the original problems inherent in the microcomputer field — a lack of inexpensive software — has apparently been alleviated.

Environment of CP/M

The essential structure of the CP/M operating system is shown in Fig. 1. The CCP is the Console Command Processor — the part of the operating system with which a user converses. A wide variety of commands is available, and these commands will be discussed below.

Basic Input/Output System (BIOS) is the section of CP/M that deals with input/output commands to all peripheral devices except the floppy disks. This includes I/O to Teletype, CRT, printer, etc. A nice feature of BIOS is that the system I/O routines are available to the user through appropriate subroutine calls in assembly-language programs. This capability constitutes a powerful addition to the assembly-language arsenal.

Basic Disk Operating System (BDOS) interfaces the system with the floppy-disk peripherals. Again, these routines are available to the user, eliminating what is typically one of the trickiest aspects of assembly-language programming — that of I/O programming.

The first 100_{16} (256_{10}) bytes of memory are used primarily as a scratchpad by the system. Various system parameters, such as where to jump on a restart, are contained in this area. In addition, a fair amout of it is available to the user. In particular, the default location of the top of the stack is location FF_{16} (255_{10}).

Finally, the Transient Program Area (TPA) is the area of memory available for user programs. It comprises the bulk of memory, even in the 16K system where it is $26FF_{16}$ (9983_{10} or 9.75K) bytes in length. In addition to user programs, all CCP transient commands are executed in the TPA. Thus, all user and most service programs originate at location 100_{16}.

The CP/M system is a disk-based system, so that an important part of the environment confronting the user has to do with the way the diskettes are structured. The diskettes are composed logically of 77 concentric tracks numbered from outside to inside as track 0 through track 76. The first two tracks (0 and 1) are used to hold the CP/M system, which is bootstrapped into memory as indicated in Fig. 1, when a cold-start procedure is initiated. Tracks 2 through 76 are available for the directory (usually on track 2) and user or system disk files (programs or data files). Each track contains 26 sectors, each of which is capable of holding 128 bytes of information. Total disk capacity, then, is a little in excess of 250K bytes, of which just over 240K is available for user files.

There are several important points to make about the disk environment. First, it

File Type	Meaning
BAS	BASIC program.
ASM	Assembly program.
SUB	Submit file.
INT	Intermediate BASIC.
PRN	Assembly results.
HEX	Assembly output.
COM	Command file.

Fig. 2. File types.

```
A>ED BIG.BAS (CR)
*100A (CR) (bring first 100 lines into buffer)
*(edit first 100 lines)
*100W (CR) (write edited lines to temporary area)
*100A (CR) (bring second 100 lines in)
*(edit second 100 lines)
*E (CR) (end edit)
```

Example 2.

```
A>PIP X.ASM = MAIN.ASM,SUB1.ASM,SUB2.ASM,SUB3.ASM
```

Example 3.

```
A>PIP TEST.BAS = CON:, X.BAS, Y.BAS
```

Example 4a.

```
A>PIP LST: = ONE.ASM, TWO.ASM, THR.ASM.
```

Example 4b.

demonstration purposes that the BASIC program shown in Fig. 4 is to be edited. We wish to insert the following line:

```
30 INPUT X
```

and also to replace line 50 by the line

```
50 NEXT I
```

These edit lines accomplish these functions:

```
*BF40↑Z-2C30 INPUT X
*BFX2YP↑ZOLKI50 NEXT I
```

The first edit line positions the pointer after the 40 and then moves it back two characters so that it is before 40. Then the INPUT statement is inserted. The second example positions the pointer after the erroneous string X2YP, then moves it to the beginning of the line, kills the line and inserts 50 NEXT I. Thus, line replacement is done by first deleting, then inserting the new line. Actually, using this editor does grow on you after some practice, even though it seems complicated at first.

As the icing on the cake, ED has several additional editing commands. For instance, the edit line

```
*BMSFIRST↑ZSECOND↑ZOTT
```

does a search for all occurrences of the string FIRST, replaces each occurrence with the string SECOND and prints out each altered line. The only new editing characters in this line are the S, which is the search command, and the M, which indicates that the next commands are to be repeated as many times as possible, i.e., until the end of the file. If a user is careful, he can perform most desired edits with the Search (S) command.

```
10 S = 0
20 FOR I = 1 TO 10
40 S = S + X
50 X2YP
60 PRINT S/10
70 END
```

Fig. 4. Sample text.

The above discussion assumes that the file to be edited is located in a memory buffer. The designers of ED were aware, however, that a particular user might not have enough memory to hold an entire program at once. Thus, the editor contains commands that have to do with bringing parts of the file into memory and writing already edited sections to a temporary disk file to make room for another segment of the unedited source. Fig. 5 gives a list of some of these commands. A user with enough memory to hold only 100 lines of BASIC might perform the sequence in Example 2 to edit a 200-line program.

Note that the end edit (E) command does several things. First, it appends any remaining lines in the memory buffer to the temporary file, then it appends any remaining source file lines to the temporary file. Next, it renames the original file, giving it file type BAK (BIG.BAK) for backup purposes. Finally, it creates a file under the original name (BIG.BAS) from the edited temporary file. So part of every editing run is a backup of the original file.

Peripheral Interface Program (PIP)

A second major transient command is the PIP program. PIP consists of a number of parts that perform utility functions for the user. One of the basic utility functions available allows the user to make a copy of an existing file. The PIP statement

```
A>PIP NEW.BAS = OLD.BAS
```

will copy the file OLD.BAS on the default disk to a new file called NEW.BAS. A rather interesting extension of this basic idea is to copy several files back to back to a newly created file. The statement in Example 3 could be used to create a program file from a main program (MAIN.ASM) and append three subroutines (SUB1.ASM, SUB2.ASM, SUB3.ASM) called by the main program. This makes a modular approach to programming easy to implement. Just save commonly used subroutines as separate files and, when needed, append them to the main program with PIP.

In order to understand the final application of PIP, we must recall the difference between logical and physical devices. A physical device is just what you would think — a TTY, a CRT, a printer, etc. Logical devices are devices defined in the BIOS, such as CON (console) and LST (list). Logical devices must be assigned to specific physical devices before communication between them is possible. Consequently, the cold-start procedure would be to assign CON to your TTY or CRT and to assign LST to your TTY (normally you would want hard-copy output). This assignment is accomplished by the use of a set of eight front-panel switches called the IOBYTE switches. For example, the switch settings

```
00000001
  ↑   ↑
 LST CON
```

accomplish this assignment. The switches not used in this example are for assigning a tape reader and punch; so unless you have such devices, these would always be left in the zero position.

PIP allows the user to refer to these logical devices, and therefore to the corresponding physical devices. If we decided to write a program called TEST.BAS, consisting of a main program to be typed in at the console that calls two subroutines X.BAS and Y.BAS already located on the default disk, we could use Example 4a.

If we simply wanted a listing of ONE.ASM, TWO.ASM and THR.ASM, we could use Example 4b. The colon is necessary to distinguish the logical device name from a disk file.

System Creation and Maintenance

CP/M contains the software necessary for procreating itself in various forms. A system can be created to accommodate any amount of memory from 16K to 64K bytes in increments of 8K. The transient commands CPM and SYSGEN are necessary to accomplish a change of system size, while SYSGEN alone will make a copy of an existing system. Typically, a user who has just installed a third 8K memory board would use the command CPM 24 * to generate a 24K system. The SYSGEN command is then used to write the newly generated

is not necessary for the user to specify where on a diskette a particular file will go. The (BDOS) system automatically finds the necessary space and keeps a record of the name and location of each file in the diskette directory. This saves the user the trouble of remembering where a particular file is located. Names of disk files are made up of three parts:

1. The first letter is used to indicate which drive (A or B) the diskette is on. This letter is optional if the operating system is told to assume that all files are on a particular drive.
2. The second part of the name is called the file name. It consists of from one to eight letters and/or numbers.
3. The last part is the file type. File type is used to indicate whether a file is a BASIC program (BAS), an assembly-language program (ASM), etc. A list of file types is given in Fig. 2.

Valid disk file names are shown in Example 1.

A file name as defined above constitutes an unambiguous file reference. In many cases, it is desirable to refer to a whole set of files with similar characteristics. This is done through the use of an *ambiguous file reference*. File references can be ambiguous in one of two ways:

1. An asterisk can be used in place of either file name or file type to indicate any file name or file type. Thus, *.BAS refers to all BASIC language source files while MYPROG.* refers to all files named MYPROG, no matter what type they are.
2. One or more question marks can be used in place of characters in either file name or file type to indicate that any character in that position is acceptable. Thus, files TEST1.BAS, TEST2.BAS and TEST3.BAS could be referred to as TEST?.BAS.

```
A:MYPROG.BAS
B:F12.ASM
PIP.COM (uses default drive)
```

Example 1.

BIOS/BDOS $3100_{16} - 31FF_{16}$
CCP $2800_{16} - 30FF_{16}$
TPA $100_{16} - 27FF_{16}$
System Area $0 - FF_{16}$

Fig. 1. Structure of CP/M 16K system.

Console Command Processor

As stated above, the CCP is the part of CP/M with which a user communicates. CCP prompts the user with a letter that indicates from which disk drive the system has been taken (also the default disk drive for file references) followed by a greater-than character (i.e., A> or B>).

Two types of commands are possible in CCP. There are built-in commands such as DIR (list directory of default disk), ERA (erase a file), REN (rename a file), TYPE (list a file) and SAVE. These commands are referred to as built in since the code for them is in the CCP area. The DIR and ERA commands allow the use of the full range of file references. For example:

```
DIR *.BAS
```

would list the names of all directory entries on the default drive that have file type BAS.

Transient commands execute in the TPA just as user programs do. A nice feature of CP/M is that, in order to execute any program (system or user), the user merely types its name in response to a CCP prompt. Thus, the runnable version of a program has a file type COM (for command).

There are five important areas addressed by CCP transient commands:

1. Program entry and editing.
2. Utilities such as copying a file from one disk to another.
3. Generating and saving various versions of the operating system.
4. Debugging aids.
5. Language processing.

We will discuss these areas one at a time.

Command	Action
B	Moves pointer to beginning of file.
-B	Moves pointer to end of file.
±nC	Moves pointer ±n characters.
nFxxx	Places pointer after nth recurrence of string xxx.
±nL	Moves pointer up or down n lines. 0L places the pointer at the beginning of a line.

Fig. 3a. Pointer positioning commands.

Command	Action
±nD	Delete ±n characters.
I	Insert text.
±nK	Kill (delete) ±n lines.
±nT	Type ±n lines on console. 0T types line up to pointer. 1T (or T) types line from pointer to end.

Fig. 3b. Basic edits.

Entry and Editing (ED)

A powerful editor (ED) is included in the CP/M operating system. This is a character editor as opposed to a line editor, meaning that a file is considered to consist of one long string of characters with CR (carriage return) and LF (line feed) characters separating each logical line. This string is held in a buffer area in memory. A pointer must be properly positioned in the text to indicate the location of each edit. Some of the basic pointer manipulation commands are shown in Fig. 3a. As an example of the use of these editing commands, the command

```
*B2FXYZ↑Z-3C
```

accomplishes the following:

- Moves pointer to beginning of buffer.
- Positions the pointer immediately after the second occurrence of the string XYZ. Note, ↑Z (control-Z) is used to delimit the string.
- Moves the pointer back three characters, i.e., it is now positioned before the X in the second occurrence (XYZ).

Once the pointer is positioned, a number of edits can be performed. These are listed in Fig. 3b. As an example of the use of these commands, consider the following two edit lines that are equivalent:

```
*B2FXYZ↑Z-3DOTT
*B2FXYZ↑Z-3C3DOTT
```

The first line positions the pointer immediately after the second occurrence of the string XYZ; deletes the three characters preceding the pointer, i.e., the XYZ; and finally prints out the resulting line. The second example first positions the pointer following the second occurrence of XYZ, then moves the pointer back three spaces, finally deletes the XYZ and prints out the resulting line.

Since programs are line oriented, it is useful to be able to perform the basic functions of a line-oriented editor: inserting a line between two existing lines, deleting a line and replacing a line. While these functions are not entirely obvious, they can be accomplished. Assume for

Command	Action
nA	Append next n lines of source to memory buffer area. n = # implies whole program.
E	End edit run. Create edited file.
Q	Quit edit run. Make no changes to file.
nW	Write n lines from memory buffer to temporary work file.

Fig. 5. Text movement editor commands.

Option	Meaning
A	Enter assembly-language mnemonics.
D	Display memory.
G	Execute with breakpoints.
L	Disassemble
M	Move a segment of memory.
S	Change memory values.
T	Trace program execution.
X	Examine CPU state.

Fig. 6. DDT command types.

system onto the first two tracks of the disk in drive B. This system can be given control by placing it in drive A and doing a restart. Thus, it is relatively easy to change system size. Even if the user has only one disk drive, this can be accomplished by modifying the SYSGEN command to write its output to drive A instead of drive B. Exactly how this is done is part of the next subject.

Dynamic Debugging Tool (DDT)

One of the more surprising transient commands to be found in the CP/M system is DDT. This command has a variety of options that enable the user to interactively execute an assembly-language program. Included in the package is the ability to set breakpoints, single or multiple step through the program, alter the command (runnable) version of a program, disassemble the command version of a program, insert assembly-language statements, examine status flags and more. These capabilities make DDT a useful and powerful part of CP/M. Fig. 6 gives a partial listing of DDT command types.

The customary process for using DDT is first to write and assemble a program so that the command version (file type COM) is available to DDT. The debugging package is then invoked as follows:

```
A>DDT TEST.COM (CR)
```

This command loads DDT into memory instead of CCP, and DDT in turn loads TEST.COM at location 100_{16}. Now any of the command types can be executed. For example, suppose we desire to test the code shown in Fig. 7a, which writes a 2 out to the front-panel programmed output lights. The assembled version is shown in Fig. 7b. Given that we have invoked DDT, we can illustrate its capabilities with a few examples. First, let's check to see if the program is in memory beginning at location 100_{16}. This is done in Example 5a and this agrees with the machine-language version in Fig. 7b.

```
JFS   ORG  100H
      MVI  A,2
      OUT  0FFH
      JMP  0
      END  JFS
```

Fig. 7a. Sample assembly-language program.

Location	Machine Language
0100	3E02
0102	D3FF
0104	C30000

Fig. 7b. Machine-language version of TEST program.

Now let's single step through the program to see if it performs its intended function (See Example 5b). The Trace command gives the state of the CPU, as indicated by the carry (C), zero (Z), minus (M), even parity (E) and auxiliary carry flags (I), the contents of the registers (A, B-C, D-E, H-L); the contents of the stack pointer (S), the program counter (P); the mnemonics of the instruction at the location pointed to by P (i.e., the instruction to be executed next); and, finally, the location from which the following instruction will be taken (0102_{16}). Let's take another step in Example 5c.

Here, the MVI A,2 instruction has been executed so the A register is changed accordingly. None of the status flags have changed. The instruction about to be executed is the OUT, OFFH, and the next instruction will come from 0104_{16}. To finish the program, one more step (Example 5d) is required. Here, the program returns control to the operating system via JMP 0. The program seems to work properly.

Two examples of the other command types are as follows:

```
-L100,106 (CR)
0100 MVI A,02
0102 OUT FF
0104 JMP 0000
```

The disassemble command recreates the assembly-language mnemonics.

```
-A100 (CR)
0100 MVI A,01 (CR)
0102 (CR)
```

This sequence replaces the MVI instruction by MVI A,1. Assembly-language statements can thus be entered at any location in the program. DDT takes care of assembling such statements. Finally, the

```
-D100,106
0100 3E 02 D3 FF C3 00 00
```

Example 5a.

```
-T (CR) (trace one step)
COZOMOEOIO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI A,02 *0102
```

Example 5b.

```
-T (CR)
COZOMOEOIO A=02 B=0000 D=0000 H=0000 S=0100 P=0102 OUT FF *0104
```

Example 5c.

```
-T (CR)
COZOMOEOIO A=00 B=0000 D=0000 H=0000 S=0100 P=0105 JMP 0000 *0000
```

Example 5d.

following sequence changes back to the original MVI instruction by changing the 01_{16} in location 101_{16} to an 02_{16}.

```
-S100
0101 01 02 (CR)
0102 D3 -(CR)
```

Note that the period ends the substitute mode.

As is readily apparent, DDT offers an invaluable tool for debugging assembly-language programs.

The Language Processors

The two main languages supported by CP/M are 8080 assembly language and BASIC. The assembler (ASM) interacts with CP/M as follows. Utilizing the editor, the user creates an assembly source file, say TEST.ASM, on disk. This file is assembled via the transient command ASM TEST. There are two important outputs of the assembly process. The results of the assembly, including error messages, are placed into a file named TEST.PRN. These results can be viewed via the TYPE TEST.PRN command. The other output is a disk file named TEST.HEX, which contains the machine-language output of the assembly. The LOAD TEST command now is invoked to create a new file named TEST.COM, which contains the binary (runnable) version of the program. This version of the program can be tested simply by typing its name as a CCP command or via DDT as described above.

```
ED $1.ASM
ERA $1.BAK
ASM $1
TYPE $1.PRN
ERA $1.PRN
LOAD $1
$1
```

Fig. 8. SUBMIT file for editing, assembly and tests.

This whole process of editing, assembling, loading and running is such a common sequence that it would be helpful to be able to teach CP/M to do the whole sequence by itself. In fact, this can be accomplished using the concept of SUBMIT files. A SUBMIT file is a disk file of CCP commands, except that the specific names (or name) of the parameters are left unspecified. Instead, they are represented by $1, $2, etc. Fig. 8 shows a listing of a SUBMIT file named AS.SUB that is useful for the above editing, assembly and test process. To instruct CP/M to execute this SUBMIT file, the user simply types the transient command

```
A>SUBMIT AS TEST
```

All occurrences of $1 are replaced by the first parameter in the parameter list (TEST), and CPM executes the list of commands as though they had been typed individually. SUBMIT files give CP/M a capability similar in nature to the job-stream concept in larger machines.

The CP/M BASIC is a full version of BASIC with floating-point arithmetic and the full complement of built-in numeric and character-handling functions. It takes 20K to run the BASIC language processors.

The procedure for running a BASIC program is to first create a disk file of BASIC source statements, say TEST.BAS. The BASIC-E TEST transient command does a partial compilation of the source file, producing an intermediate file called TEST.INT. The RUN-E TEST command is used to load and run the program.

Conclusion

The intention of this article has been to present enough details of the CP/M operating system to give the reader a flavor for the degree of sophistication of currently available software.

It is an interesting intellectual exercise to think about writing one's own operating system, but it seems clear that with such sophisticated software available at a reasonable price, the time and cost of writing an operating system is prohibitive.

No comparison has been attempted between CP/M and similar software products on the market. It's difficult enough to keep track of the names of all the companies dealing in various aspects of the microcomputer market. It would take a great deal of time to evaluate all the software competitive with CP/M. I hope this article will offer a friendly challenge to others knowledgeable in particular micro operating systems. Let's see an article or two on these other systems. Let's bring micro-systems software out in the open. The personal effort is worthwhile and would be instructive to us all. ■

References:

Imsai CP/M Floppy Disk Operation System Version 1.31, Rev. 0, 1976, Imsai Manufacturing Corporation, San Leandro CA 94577.

An Introduction To CP/M Features and Facilities, 1976 (this and all following refs. by Digital Research, Pacific Grove CA 93950.

ED - a Context Editor for the CP/M Disk System, User's Manual, 1976.

CP/M Assembler (ASM) User's Guide, 1976.

CP/M Interface Guide, 1976.

CP/M System Alteration Guide, 1976.